Program Schemas with Tree Data Structures (アルゴリズムにおける証明論)

AUTHOR(S):

TOKURA, NOBUKI; KASAMI, TADAO

数理解析研究所講究録
236巻 1975年 197-203

PROGRAM SCHEMAS WITH TREE DATA STRUCTURES

Nobuki TOKURA, Tadao KASAMI

Faculty of Engrg. Sci. Osaka University, Toyonaka

1. *Introduction*

Conventional program schema [1][4][6][10] may be thought of as a model for those programs which perform very deep computations on a small number of data. On the other hand, there are another type of programs, such as control programs, which process a large number of data of some data structure and perform simple computations on each data item. We have analyzed a small disk operating system (DOS) monitor program [2][3]. The whole data structure in the DOS monitor is rather complicated. But, each routine (or module) can be said to process a rather simple substructure such as a linear list. In this paper, some program schemas with data structures are introduced as models for programs of the latter type, and it is shown that the equivalence problems of these program schemas are decidable. Constable, Gries and Chandra [6][10] investigate program schemas with data structures such as arrays and pushdown lists. In their model, data structures are used as a kind of working strages. Their program schemas are so powerful that most decision problems are undecidable. Their works mainly deal with the problems of translation from one class of schemas to another class. On the other hand, our program schemas have tree data structures as both inputs and outputs. And by some restriction, their powers remain in the region such that their equivalence problems are decidable.

2. *Definitions*

The program schema which we consider is defined as follows by using BNF

notation:

<program> :: = (a {,a}) : <body>

<body> :: = <S-list> ; [ℓ:] HALT (z)

<S-list> :: = [ℓ:] <S> {; [ℓ:] <S>}

<S> :: = P ← P·up (ℓ) (1)

| P ← P·down i (ℓ) (2)

| q (P, a_1,···, a_{Rq}) ℓ_1, ℓ_2 (3)

| q (R) ℓ_1, ℓ_2 (4)

| Output f (P, a_1,···, a_{Rf}) (5)

| R ← f (P, a_1,···, a_{Rf}) (6)

| R ← f (R, P, a_1,···,a_{Rf}) (7)

| HALT (z) (8)

| GO TO ℓ (9)

In this definition, ℓ denotes a label from the set $L = \{L_1, L_2, \cdots\}$, a denotes an input register from the set $A = \{A_1, A_2, \cdots\}$, f denotes a basic function symbol from the set $F = \{F_1, F_2, \cdots\}$ and q denotes a predicate symbol from the set $Q = \{Q_1, Q_2, \cdots\}$. z denotes a natural number. Each predicate symbol Q_i has rank $RQ_i + 1$ for the statement of type (3) and rank 1 for the statement of type (4). Each function symbol F_i has rank $RF_i + 1$ for the statement of type (5) and (6) and rank $RF_i + 2$ for the statement of type (7). The output statement of type (5) writes the value $f(P, a_1, \cdots, a_{Rf})$ into the field of the vertex pointed to by P. R is the output register. For simplicity, we consider only one output register.

The list of input registers before the <body> indicates the input registers which are used in the schema. As usual, any label ℓ used in a GOTO statement or conditional statement of types (3) and (4) must also label a statement. Each label ℓ can be used only once to label a statement. Each program schema has only one pointer, represented as P, which scans a tree data structure. A rooted

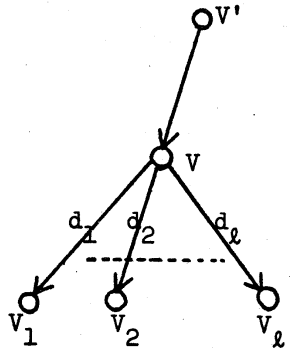

Fig. 1.

tree (or simply tree) is a directed graph satisfying the following three conditions:
(1) There is exactly one vertex, called the root, which no edge enters. (2) For each vertex in the tree there exists a sequence of directed edges from the root to the vertex. (3) Exactly one edge enters each vertex except the root. A vertex from which no edge exits is called a leaf. Consider a subtree of Fig.1. When pointer P points to the vertex V, P moves to the vertex V' by the operation P ← P•up(ℓ) (1), and P moves to the vertex V_i by the operation P ← P•down i (ℓ) (2). Value of P•up is NIL when P points to the root and value of P•down i is NIL when P points to a leaf. And if P points to the vertex V in Fig.1 and if there is no edge d_i, then P•down i takes the value NIL. If the right hand is NIL in the operations (1) or (2), then the control will jump to the statement labelled with ℓ and the pointer P is left unmoved.

Let D, the class of tree data structures (or simply, data structure) be defined as follows: It is assumed without loss of generality that each vertex has only one data field. And the size of each data field is considered not to be bounded. This covers the cases of an arbitrary number in the binary notation and symbol strings with arbitrary length. The numbers of user names and file names in the example below may be considered to be potentially infinite. In this way, we discriminate between those entities whose sizes are proper to the program and others which are treated as not necessarily bounded. Although in real programming languages, there might be some bounds for the entities of the latter type, algorithms which depends on the boundedness of this kind might be too inefficient to carry out. In this paper, only data fields are modified and the data structure remains unchanged. (Note. in Remark 2, a schema which

changes the structure to a certain degree is mentioned.)

Example 1: Let us consider a data structure called a file directory in Fig.2 and the following program schema S:

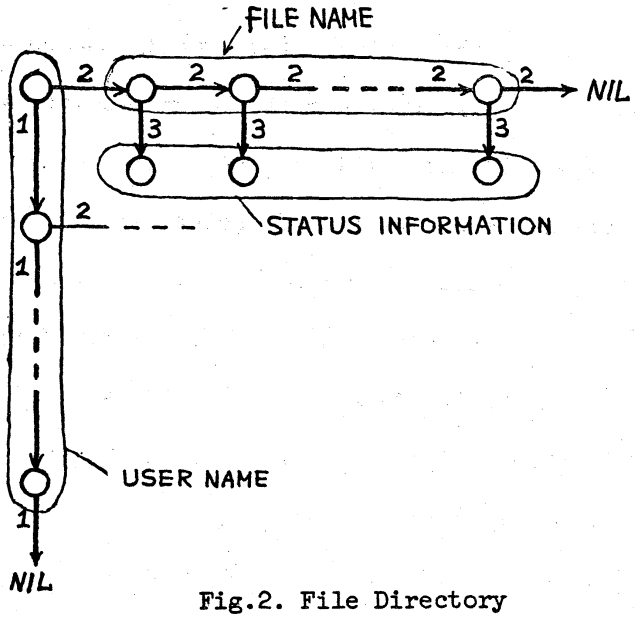

Fig.2. File Directory

(A_1, A_2) : L_0 : Q_1 (P, A_1) L_2, L_1

L_1 : $P \leftarrow P \cdot down\ 1\ (L_5)$;

GO TO L_0 ;

L_2 : $P \leftarrow P \cdot down\ 2\ (L_4)$;

Q_1 (P, A_2) L_3, L_2

L_3 : $P \leftarrow P \cdot down\ 3\ (L_6)$;

$R \leftarrow F_1$ (P) ;

HALT (3) ;

L_4 : HALT (2) ;

L_5 : HALT (1) ;

L_6 : HALT (0)

where F_1 and Q_1 are uninterpreted. If an interpretation

$Q_1(X, Y)$: $X = Y$

$F_1(X)$: $F_1(X) = X$ (identity function)

is given, then the program schema becomes a program such that the status information of the file given by A_2 of the user given by A_1 in a file directory is returned to the register R. Then, the halting condition may be described as follows :

HALT (0) : the status information is not found.

HALT (1) : the user is not registered.

HALT (2) : the file is not found.

HALT (3) : the status information is in R.

The HALT statement of type (8) gives a decision as above. This class of program schema can model many of the modules of monitor programs and others.

3. Program schemas with tree data structures

1° The execution of the program starts with its pointer pointing at the root. It is assumed without loss of generality that HALT statements will be executed only after P ← P·up is executed with the value of P·up being NIL, that is, HALT statement is executed when the pointer moves off the root upwards.

2° Two program schemas S_1 and S_2 are said to be equivalent if under any interpretation, for any input data structure of D, either both of them do not halt or both of them halt by executing HALT(i) for some i in such a way that the contents of all the corresponding fields of the resulting data structures are identical and the contents of the output registers are identical.

3° Class 1 of program schemas —— PS_1

Let program schemas of PS_1 be those which satisfy the following condition:

a1) Each vertex is scanned not more than K times for a fixed K and output statements of type (5) are not executed at each vertex more than L times for a fixed L under any interpretation.

Theorem 1: It is decidable whether two program schemas of class PS_1 with no use of the statement of type (6) are (strongly) equivalent.

Theorem 2: It is decidable whether two program schemas of class PS_1 with no use of the statement of type (4) are equivalent.

Remark 1: If the assumption a1 is removed, then we have negative results. More strongly, it can be shown that it is recursively undecidable whether two Turing acceptors are equivalent which are permitted to scan the (one-dimensional) tape indefinitely but are permitted to rewrite each square only once.[11]

4° Class 2 of program schemas —— PS_2

Let program schemas of PS_2 be those which satisfy the following condition:

a2) It is assumed that each vertex has two disjoint data fields. One of them is the output field of the output statement of type (5) and the other is a field which are referenced to by the statements of type (3), (5) and (6).

This condition is introduced in consideration of Remark 1. The schema of PS_2 may loop on a tree structure under some interpretation and it does not necessarily satisfy the assumption a1).

Theorem 3: It is decidable whether two program schemas of class PS_2 with no use of the statement of type (6) are equivalent, under the assumption that the output statements are executed at most once for each vertex.

Theorem 4: It is decidable whether two program schemas of class PS_2 with no use of the statement of type (4) are equivalent.

There are two reasons for us to consider the class PS_2. First, several typical modules of monitor programs satisfy the conditions for PS_2. Second, while there are no known algorithm to decide whether a schema satisfies the assumption a1), it is decidable whether a schema satisfies the assumption a2).

Remark 2: By the same proof technique, the results can be extended to the class of program schemas which modify the tree structure to a certain degree. The following modifications are permissible.

1) To permute the outgoing edges of each vertex.

2) To delete a vertex or to delete a subtree.

3) To insert a bounded number of vertices into each edge.

The above results are proved by reducing the problem to the one of tree automaton. Two-way tree automaton (2ta), tree walking automaton (twa), tree-tape Turing machine (tTM) and others are investigated. The results are omitted here. A full paper is in preparation.

References

[1] I.I.Ianov, "The logical schemes of algorithms," Prob. of Cyb. 1 (1958), 75-127.

[2] J.Okui, N.Tokura and T.Kasami, "Analysis of a control program — a linear list processing machine," Papers of Technical Group on Automata and language, I.E.C.E., Japan (April, 1972).

[3] ———, "Analysis of a disk operating system," The second symp. on MTC, Research Inst. for Math. Sci. Kyoto Univ. 1972.

[4] M.S.Paterson, "Equivalence problems in a model of computation," Artificial Intelligence Tech. Memo, 1, M.I.T. 1970.

[5] N.Tokura and T.Kasami, "Program schemas with tree data structures," Symp. on Math. theory of inform. sci., Research Inst. for Math. Sci. Kyoto Univ. Feb. 1973.

[6] R.L.Constable and D.Gries, "On Classes of Program Schemata," SIAM J. Comput. 1 (1972), 66-118.

[7] Y.Katsuyama, K.Taniguchi, N.Tokura and T.Kasami, "Simplification of Program Schemata," Papers of Technical Group on Automata and language, I.E.C.E., Japan (Jan. 1973).

[8] J.Okui, T.Hosomi, N.Tokura and T.Kasami, "A formal definition of file management programs," Op. Cit. (Jan. 1973).

[9] T.Hosomi, N.Tokura and T.Kasami, "Some program schemata for file processing," Op. Cit. (June 1972).

[10] A.Chandra, "On the properties and application of program schemas," Stanford Artificial Intelligence Lab. MEMO AIM-188, Comp. Sci. Dept. Stanford Univ. 1973.

[11] M.Minsky, Computation : Finite and infinite machines, Prentice Hall 1967.